FPGA TOOLS – HOW TO

Bharathwaj "Bart Simpson" Muthuswamy
Connectivity Lab
264M Cory Hall
Department of EECS
University of California, Berkeley
mbharat@eecs.berkeley.edu

This document serves as a tutorial for using the myriad of software tools used to program an FPGA.

Describing the complete functionality of an FPGA is beyond the scope of this document, an excellent reference is [1]. The language that we use to program an FPGA is Verilog. Describing Verilog in detail is also beyond the scope of this document. Although you could use the examples in this report to understand the basic ideas behind Verilog, I would suggest consulting a reference like [2].

In addition, you need an FPGA hardware platform for implementing your design. As stated earlier, we use the ML403 for testing some of our designs. This tutorial assumes the implementation platform is the ML403. You should be able to adopt the steps below for any platform, shoot me an email if you are stuck and I will try to help. I will also assume that you have a good text editor (I use Crimson Editor), Verilog simulator (we use ModelSim 6.2a), FPGA programming environment (we use Xilinx ISE 9.1) and a Power PC Programming toolkit (we use Xilinx EDK 9.1). You can also program an FPGA using Simulink from Mathworks: you need System Generator for DSP (we use version 9.1). Please note: installing FPGA programming tools is a long (approximately 2 days) and error-prone process. You are better off having them installed by an expert if you are new to FPGA programming. In any case you should work through the tutorial below on a computer.

1. Functional Simulation of Verilog Code using ModelSim

ModelSim is a powerful simulator from Mentor Graphics for simulating Verilog code. The best reference to learn ModelSim is the ModelSim User's Manual which you can download from Mentor Graphics' website (after registering for free). Consider the simple Verilog code snippet below that implements a 3-bit counter. You are encouraged to make a new folder for the counter and its' test-bench.

```
module counter_3bit(clock,enable,reset,count);
	input clock,reset;
	input enable;
	output [2:0] count;

	reg [2:0] count;

	always @(posedge clock or posedge reset)
	begin
		if(enable)
```

```
        begin
            if (reset)
                count <= 3'b0;
            else if(count == 3'b111)
                count <= 3'b0;
            else
                count <= count+1;
        end
    end
endmodule
```

The code above is hopefully straightforward to understand. The code snippet below shows a testbench for the 3-bit counter:

```
`timescale 1 ns/1 ps
`define HalfCycle 5
`define Cycle (`HalfCycle * 2)

module counter_3bit_testbench;
    reg clock,enable,reset;
    wire [2:0] count;

    //setup clock
    initial clock = 1'b0; /* you need to do this since clock
                  is of type reg and by the Verilog standard,
                  reg types start with "x" value unless  explicitly set */
    always #(`HalfCycle) clock = !clock;

    //setup UUT (Unit Under Test)
    counter_3bit testcounter(
        .clock(clock),
        .enable(enable),
        .reset(reset),
        .count(count));

    initial begin
        // apply some test stimulus
        enable = 1'b1;
        reset = 1'b1;
        #(`Cycle * 3);
        reset = 1'b0;
        #(`Cycle * 5);
        reset = 1'b1;
        #(`Cycle);
        reset = 1'b0;
    end

endmodule
```

We will use ModelSim to simulate the code above. Start ModelSim by double-clicking on the 

icon. Figure 1 should pop up.

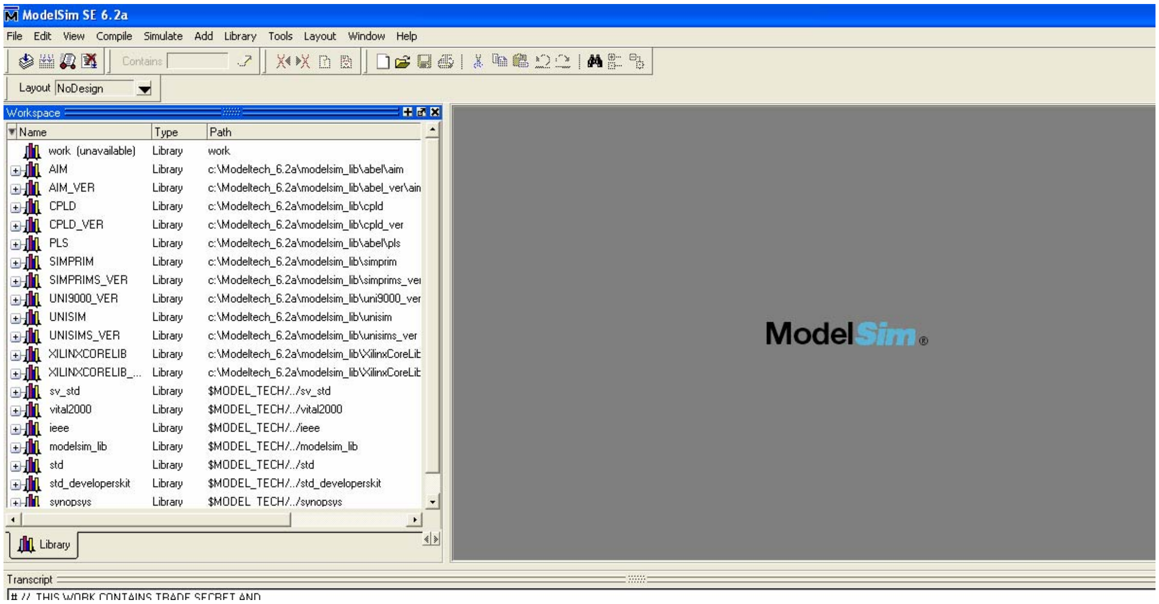


Figure 1. The ModelSim startup screen

A note on the Workspace window: in figure 1 you see a lot of libraries related to Xilinx products. You need these libraries if you are to use any of Xilinx's parts (like RAM units synthesized from Xilinx CoreGen) in your code. You won't need these parts for the 3bit counter. Here are the steps to simulate your model. What you type is in ***bold-italics*** ModelSim output is in *10-point italics*

Step 1: cd to your project directory by typing the command in the Transcript window:

ModelSim> cd sample_code/counters

Step 2: One way to use ModelSim is to create a library and add your modules to it. For other methods refer to the ModelSim User's Guide:

ModelSim> vlib work

ModelSim> vmap work work

 # Modifying C:\Modeltech_6.2a\win32/../modelsim.ini

Step 2: Compile the code:

ModelSim> vlog *.v

*# ** Note: (vlog-1901) OptionFile "C:\Modeltech_6.2a\vlog.opt" not found. Ignored.*
 # Model Technology ModelSim SE vlog 6.2a Compiler 2006.06 Jun 16 2006
 # -- Compiling module counter_3bit
 # -- Compiling module counter_3bit_testbench
#
 # Top level modules:
 #   counter_3bit_testbench

Step 3: Start the simulation:

ModelSim> vsim work.counter_3bit_testbench

 # vsim work.counter_3bit_testbench
 # Loading work.counter_3bit_testbench
 # Loading work.counter_3bit
*# ** Warning: (vsim-3009) [TSCALE] - Module 'counter_3bit' does not have a `timescale directive in effect, but previous modules do.*
 #   Region: /counter_3bit_testbench/testcounter

Figure 2 will pop up.

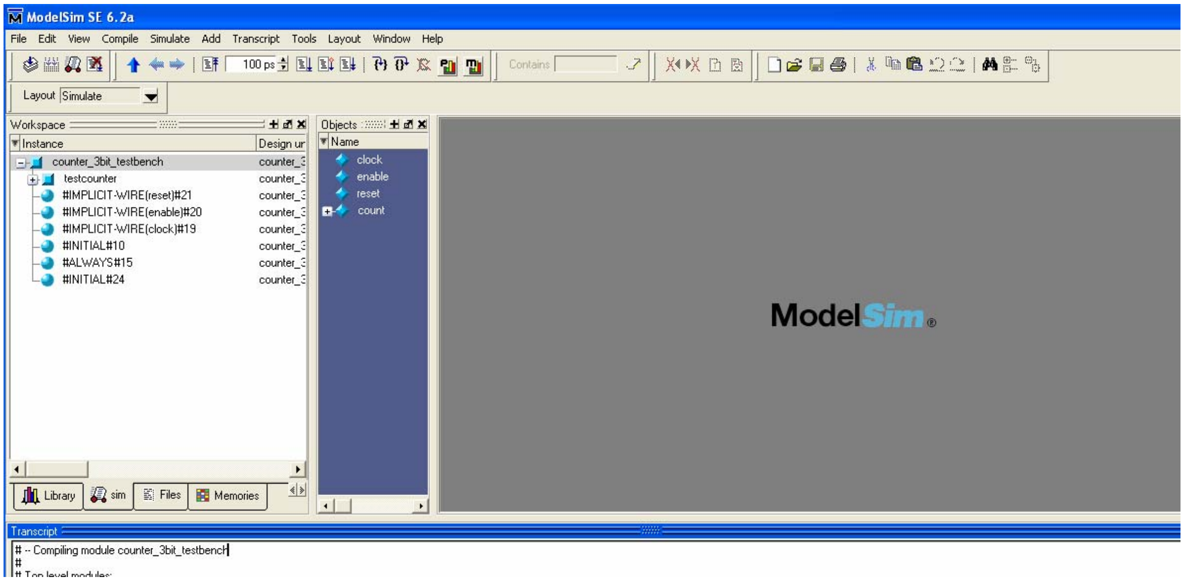


Figure 2. ModelSim all ready to simulate

Now we need to add what waveforms that we want to see. I usually add all waveforms:

Step 4: Add waveforms:

ModelSim>add wave *

Figure 3 should pop up.

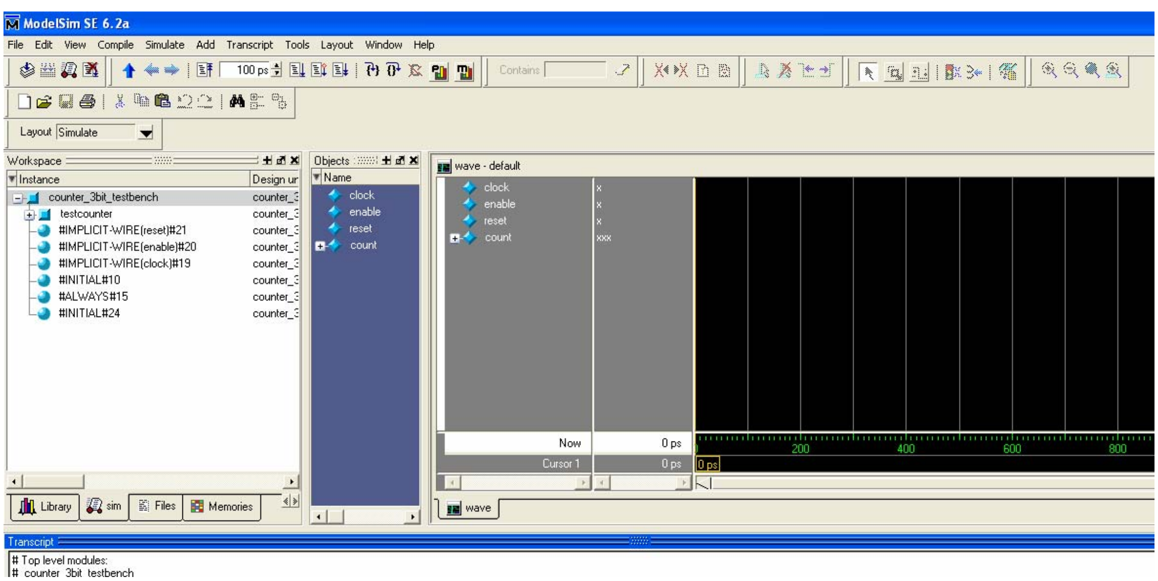


Figure 3. The wave window has been added to ModelSim

Step 5: Now run the simulation:

ModelSim> run 500ns

I am showing only the wave window in figure 4.

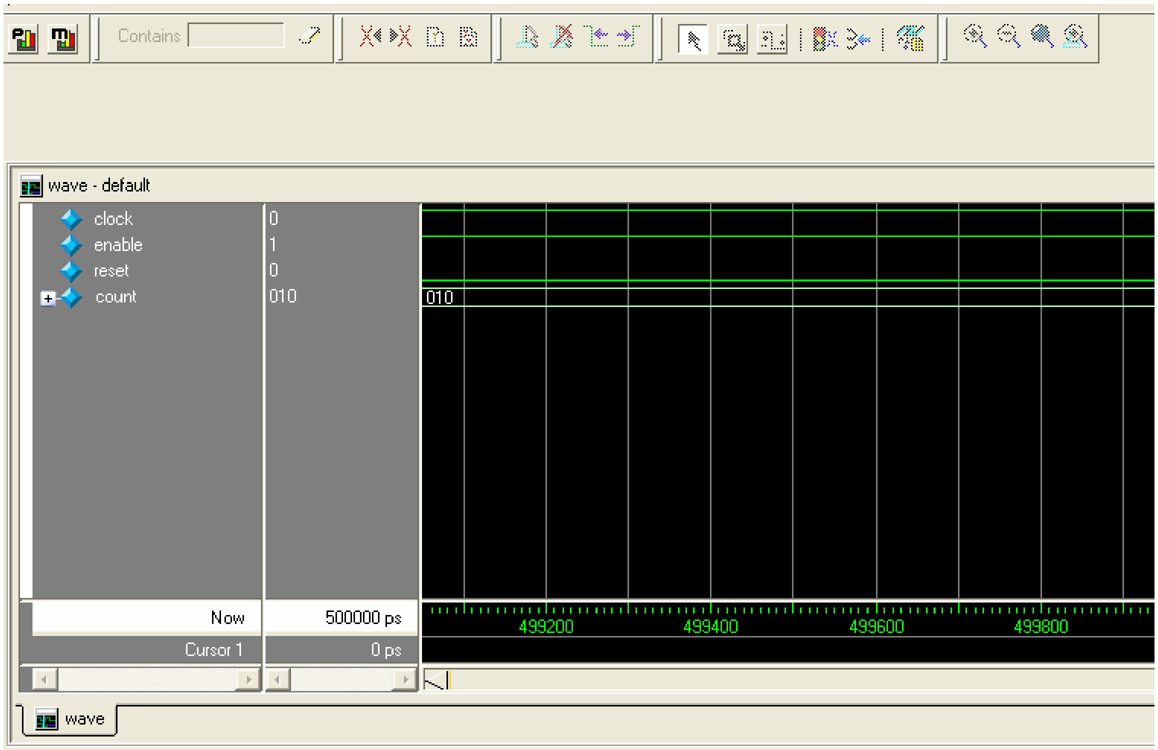
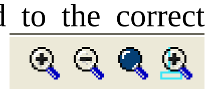
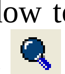


Figure 4. The wave window with our simulation results

The wave window is not zoomed to the correct resolution. First click in the wave window to make the zoom toolbar  active. Next click on the Zoom Full icon: . Figure 5 is the result.

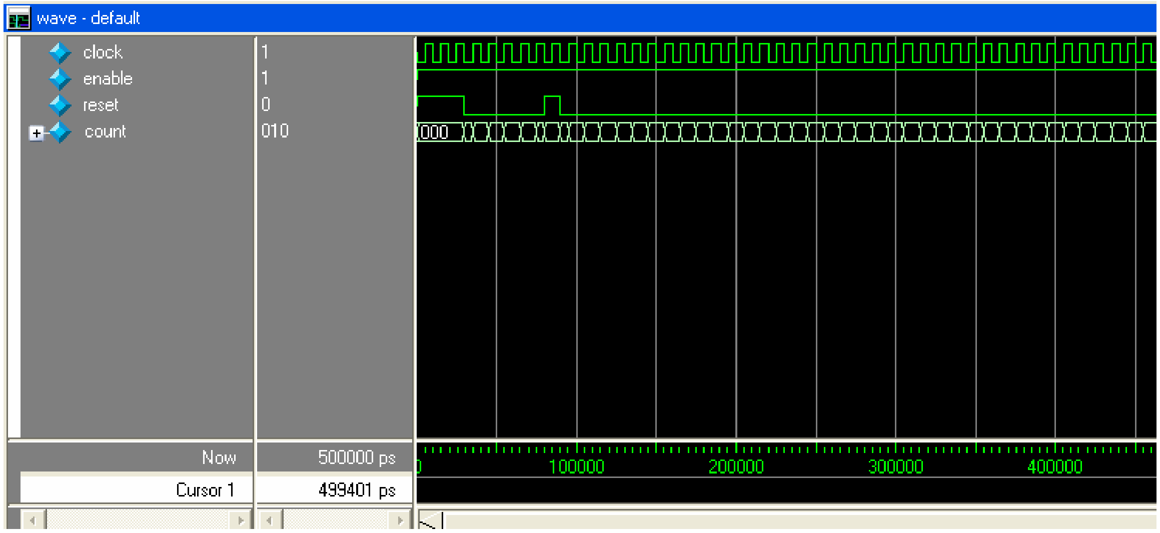


Figure 5 The wave window showing more details

You can use the other zoom icons to get a nicer picture. Try to get figure the wave window to look like figure 6. You can also use the cursor icons:  to place cursor(s) on the wave window so that you can measure properties like the rise time etc.

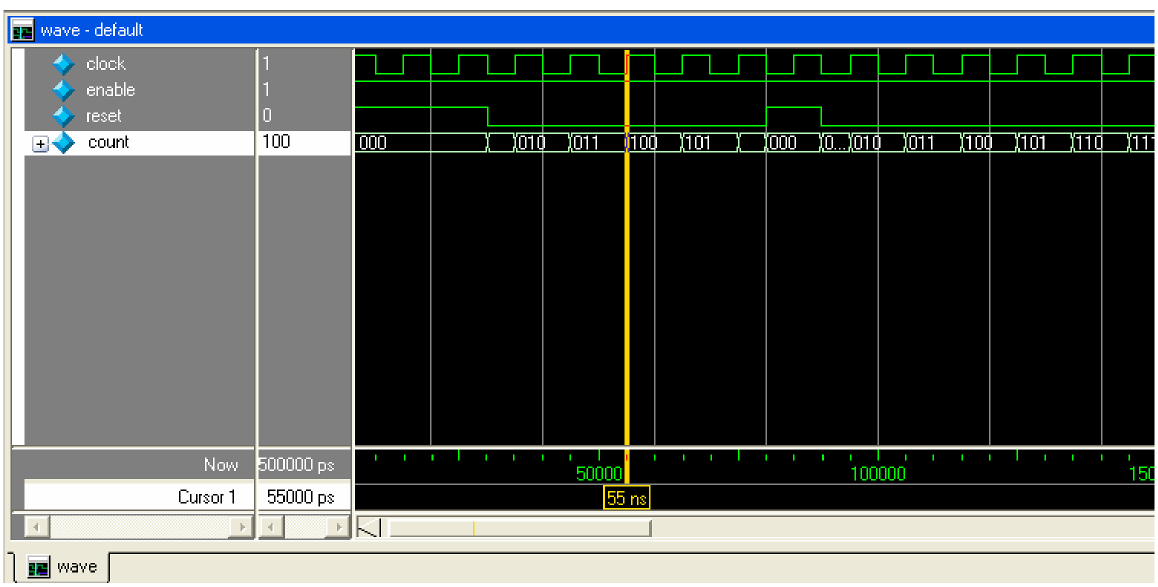


Figure 6. Wave window with an active cursor and showing more functionality of the counter.

2. Implementation of your code on an FPGA – Using Xilinx ISE to simplify the steps

We will implement the counter created in section (a) on the ML403 board. The tool we will use is Xilinx ISE. Implementation of a design on an FPGA mainly consists of the

following steps: Synthesis, Translation, Generating a Program File and Download the program file onto the FPGA.

We won't go into a lot of detail(s) on the steps above, just an idea of how to use the ISE. An important note: the ISE is a GUI: Graphical User Interface. That is, it is just an interface to command line tools. Therefore when there are errors in any stage of the ISE, you should look at the output dumps from each step. They will tell you a lot about what is going on. Lets get started:

Step 1: Double-click on the Xilinx ISE icon 

on your desktop to start the tool. Figure 7 shows up.

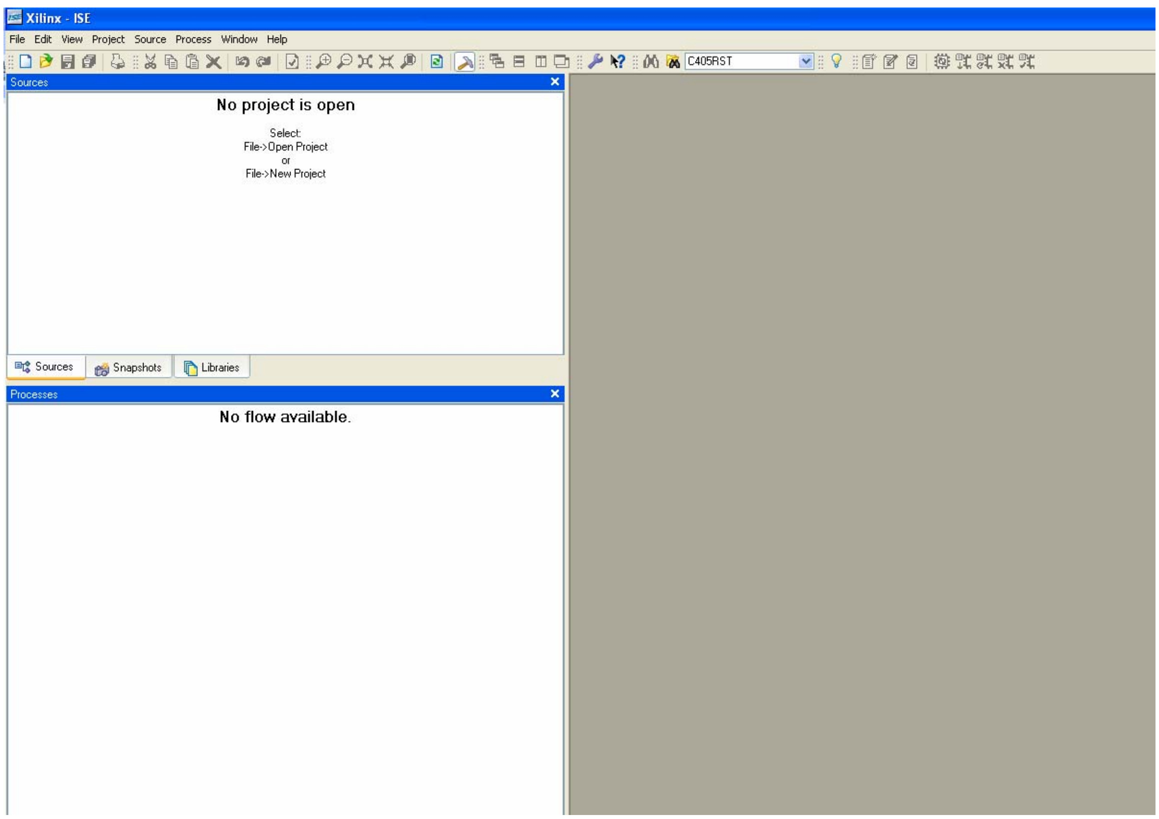


Figure 7. Xilinx ISE startup screen

Step 2: Click on ***File*** $\rightarrow$ ***New Project*** to start the New Project Wizard shown in figure 8

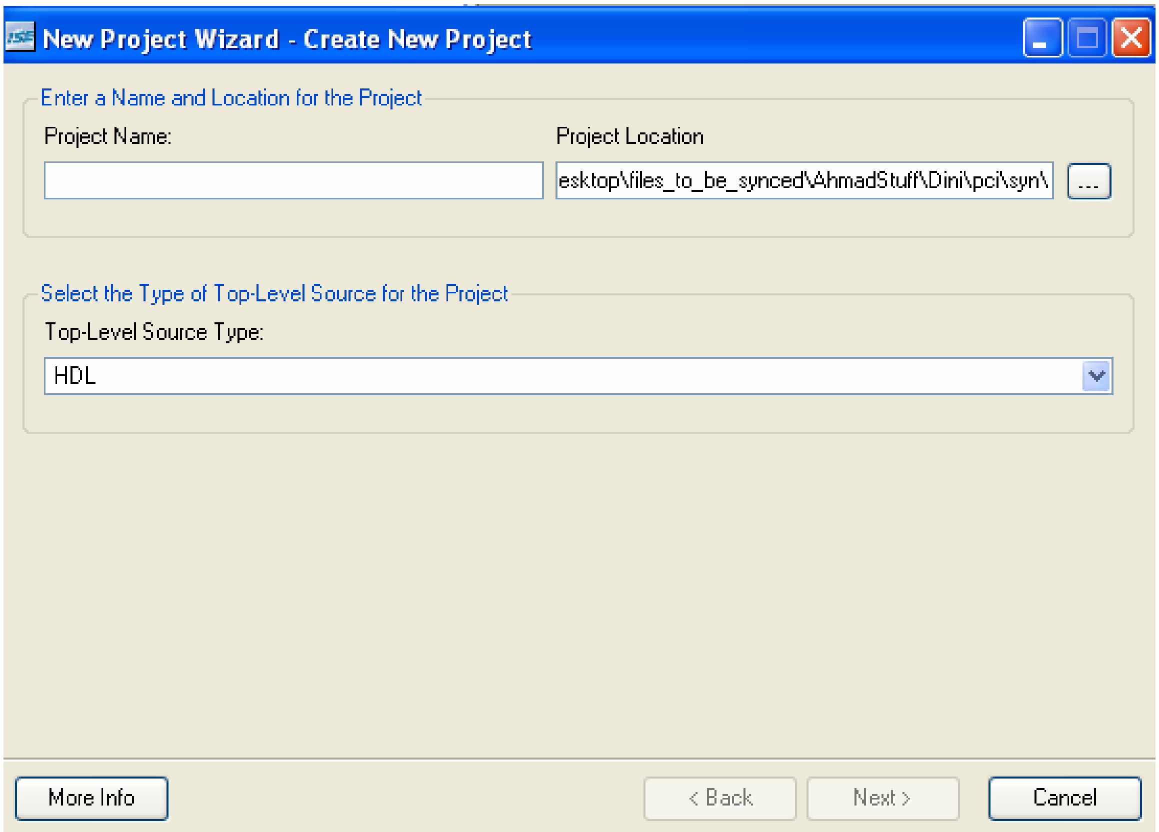
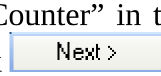


Figure 8. ISE New Project Wizard

Step 3: Create a new project named "ML403_Counter" in the same directory as your Verilog source files. Leave source as HDL. Click 

to continue.

Step 4: The Device Properties window will pop up. Configure it to match the chip on the ML403 board (see figure 9). If you have the board, just look at the chip. This will give you all the information you need.

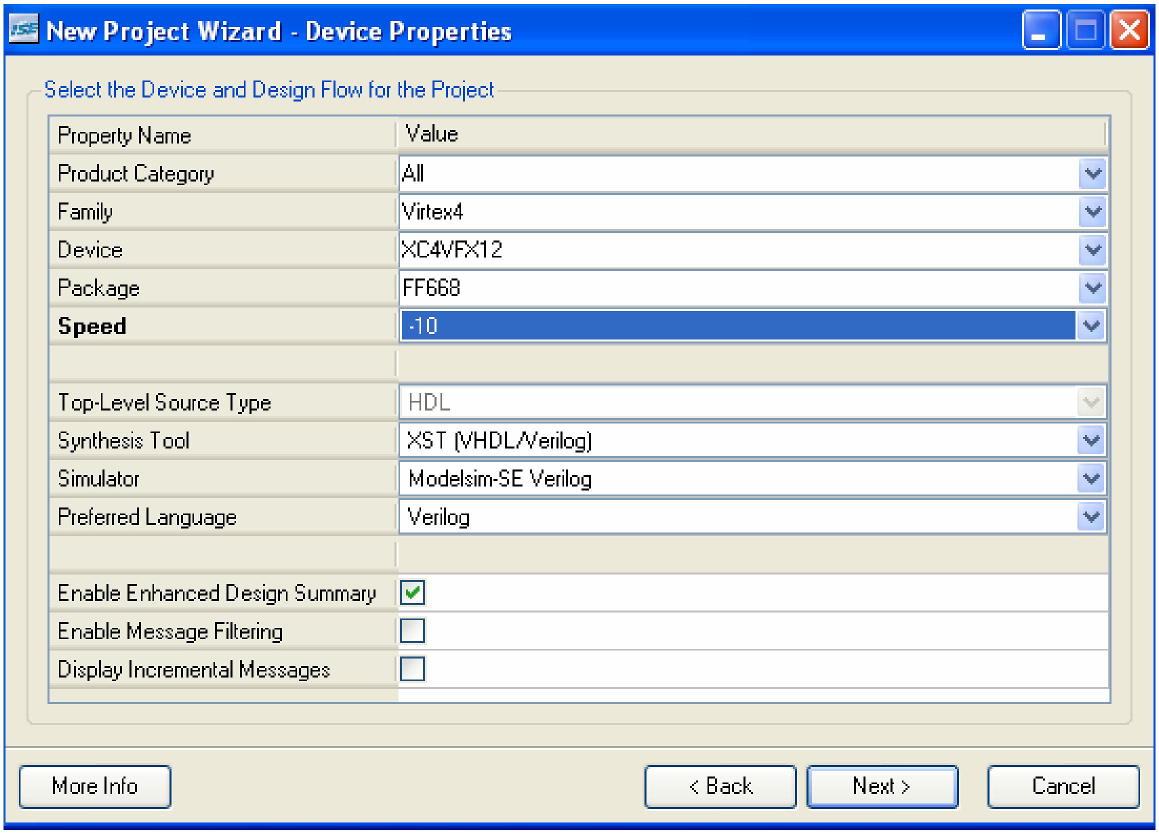


Figure 9. Device properties for the ML403

Step 5: Click Next > **twice** so that you can add an existing source. Click Add Source to add the Counter_3bit.v file. Do not add the testbench

Step 6: Accept the default project settings. Click Finish. You should see figure 10 (click OK if you get the "File Added Successfully" box)

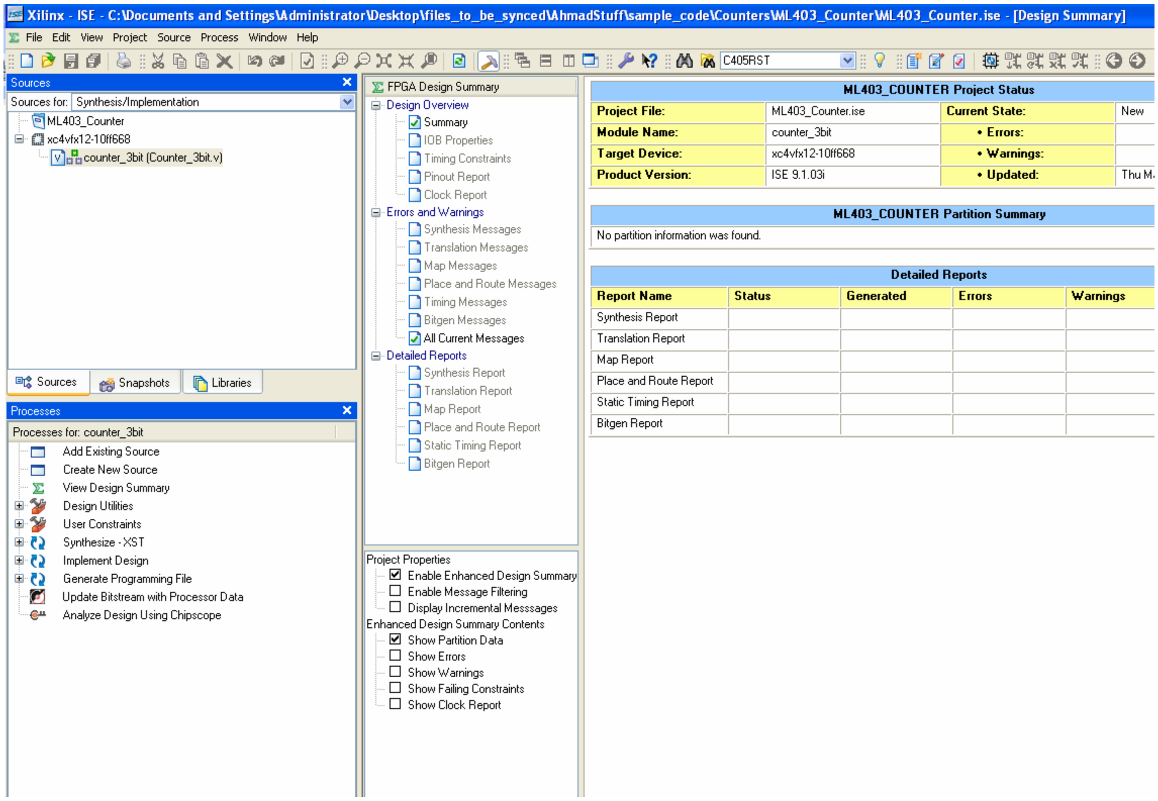


Figure 10. ISE project has been created!

Now there are two steps that you need to do before you can build your code. First you need to tell the design tools what is your top level module is. This is the module that starts executing once you download code to the board. Second you need to tie the input and outputs from your code to actual FPGA pins. There are two ways to do this: using a User-Constraints-File (UCF or XCF, Xilinx Constraints File) or directly specifying the pin locations in your top level module [3]. I will use a UCF for the pin locations, refer to [3] for directly specifying the constraints in your top level module.

Step 7: Now converting counter_3bit to a top-level module is pretty easy: you just have to specify the pin locations in the code! Double-click on the counter_3bit in the Sources dialog box (refer to figure 10) and modify the code as shown below.

```
module counter_3bit(
	clock,	// goes to FPGA pin AE14, SYSCLK (100 MHz)
	reset,	// goes to FPGA pin E7, GPIO Switch North
	/* count0 will go to FPGA pin G5, GPIO LED 0
		count1 will go to FPGA pin A6, GPIO LED 1
		count2 will go to FPGA pin A11, GPIO LED 2
	*/
	count);
	input clock,reset;
	output [2:0] count;
	reg [2:0] count;

	// we need to slow down the clock or we can't distinguish the count
	reg slow_1Hz_clock;
	reg [31:0] fast_100MHz_count;

	initial slow_1Hz_clock = 1'b0;
	initial fast_100MHz_count = 32'b0;

	/* if our 32-bit counter reaches 5e7, then
		0.5 seconds have elapsed (the input clock is at
		100 MHz = 1e8 Hz)	*/
	always @(posedge clock or posedge reset)
	begin
		if (reset)
			fast_100MHz_count <= 32'h0;
		else if(fast_100MHz_count == 32'h2FAF080)
			begin
				fast_100MHz_count <= 32'h0;
				slow_1Hz_clock = !slow_1Hz_clock;
			end
		else
			fast_100MHz_count <= fast_100MHz_count + 1;

	end

	always @(posedge slow_1Hz_clock or posedge reset)
	begin
			if (reset)
				count <= 3'b0;
			else if(count == 3'b111)
				count <= 3'b0;
			else
				count <= count+1;
	end

endmodule
```

In the code snippet above, I have done two things. First, I specified which pins on the FPGA the ports of the top level module map to. In order to find this information, refer to your board’s User’s Guide. In the case of the ML403, refer to [4]. Next I slowed down the clock to 1 Hz using a 32-bit counter.

Step 8: Next let us specify a UCF. Create a file named “counter_3bit.ucf” in the same directory as your project. Enter the following using a text editor:

```
NET "clock" LOC = "AE14";
NET "reset" LOC = "E7";
NET "count<0>" LOC = "G6";
NET "count<1>" LOC = "A11";
NET "count<2>" LOC = "A12";
```

Save the file. There are actually two types of constraints: placement and timing. However for most designs, it is un-necessary to specify timing if you specify a global clock. For further information on timing constraints, please refer to [3].

Step 9: Now add the UCF file you created in step (8) to your project. Right-click the xc4vfx12-10ff668 under sources and select Add Source as shown in figure 11.

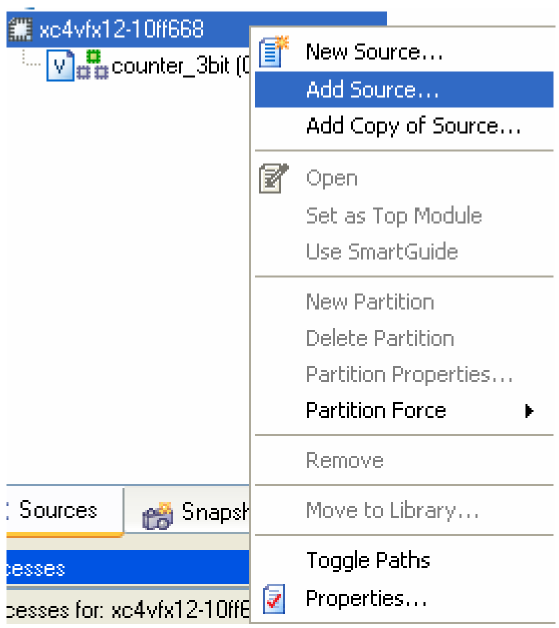


Figure 11. Adding the UCF

Step 10: Left-click the counter_3bit (Counter_3bit.v) under the Sources window as shown in figure 12. The Processes dialog box will change to reflect the implementation steps. Double click on the Generate Programming File. ISE will start the synthesis process.

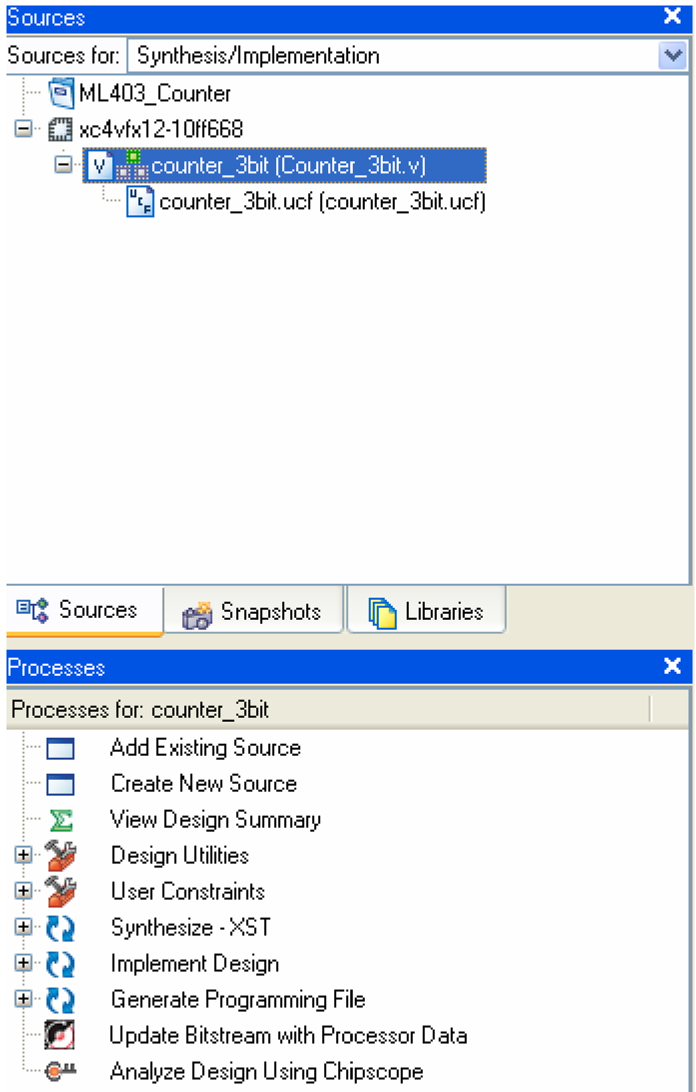


Figure 12. Double click on Generate Programming File to start bitstream generation

Step 11: If all goes well, ISE should be done in about a minute (on a dual-core 2 GHz machine). The next step is to actually download your design to the FPGA. Connect your JTAG cable to the ML403 (refer to [4]) via the "FPGA & CPU Debug" port. Power on your ML403 and connect your JTAG cable to your computer.

Step 12: Left-Click on "+" and double-click on the Configure Device (iMPACT) option. Figure 13 pops up. Leave the defaults, click Finish. Figure 14 pops up.

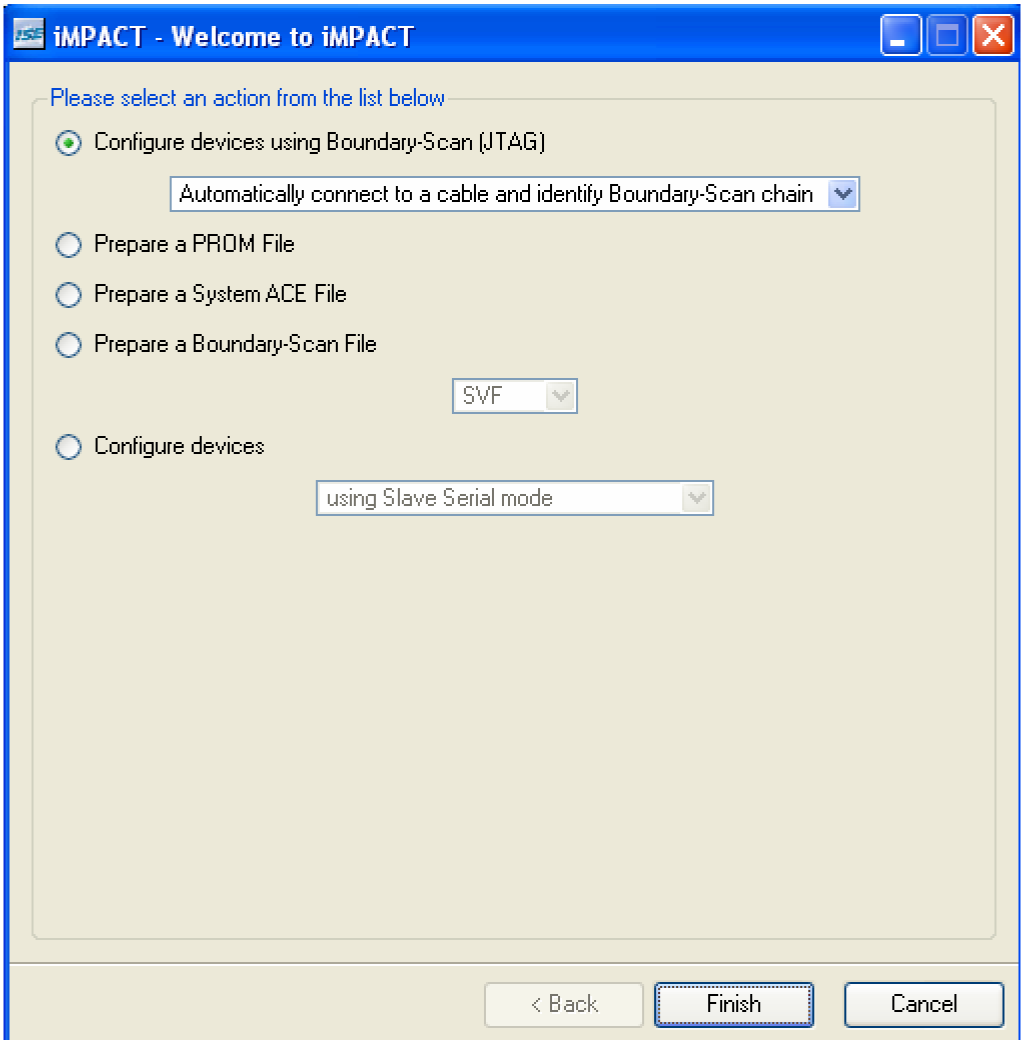


Figure 13. Selecting the cable type

Step 13: You can skip configuration of the xccace, xcf32p and the xc95144xl. All you are going to configure is the FX chip. So, click Bypass till you get to the FX chip.

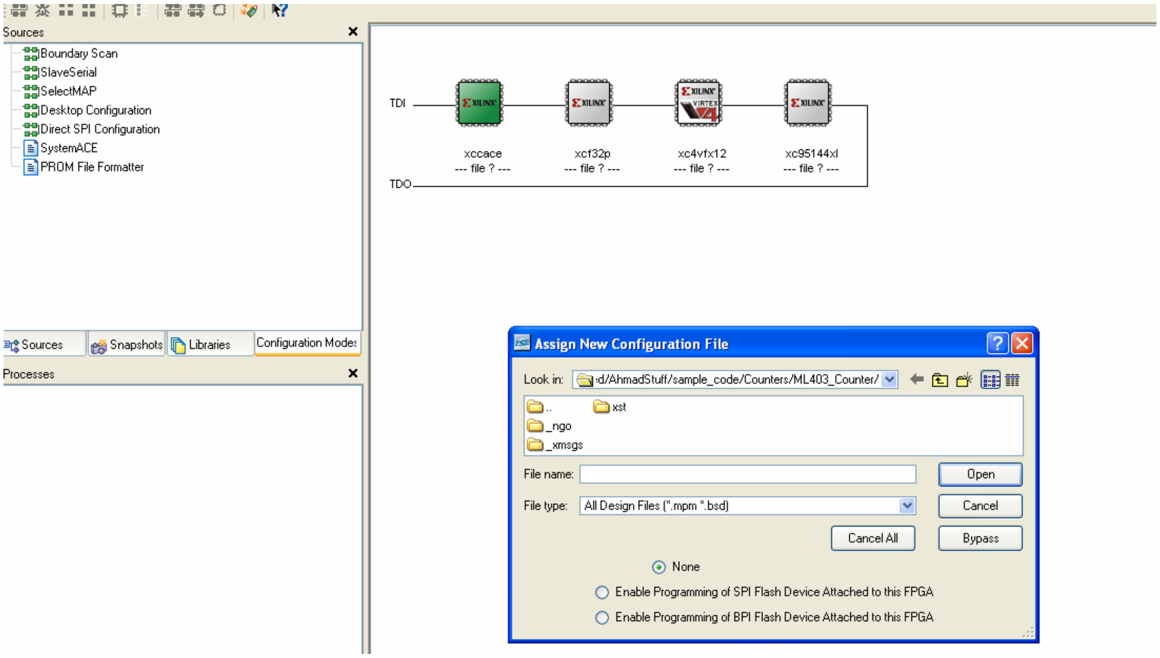


Figure 14. Getting ready to download the bit file

Step 14: Select the counter_3bit.bit file. Just select OK in the ensuing dialog box, you are not going to add any other files. Now right click on the xc4vfx12, left-click program and select OK. That's it! You should see the LEDs light up on the board.

3. Embedded Power PC Programming using Xilinx EDK

NOTE: DO NOT CREATE THIS PROJECT IN A DIRECTORY WITH SPACES! IF YOU DO, THE PROJECT WILL NOT COMPILE. THE REASON IS: XILINX TOOLS USE Cygwin CALLS, SINCE THEY ARE UNIX BASED THEY WILL NOT HANDLE SPACES!

In the previous section, you learned how to program the FPGA using ISE and HDL. In this section you will look at how to program the Power PC (PPC) core on the Xilinx FX chips. The basic concept behind programming the FX chip is shown in figure 15.

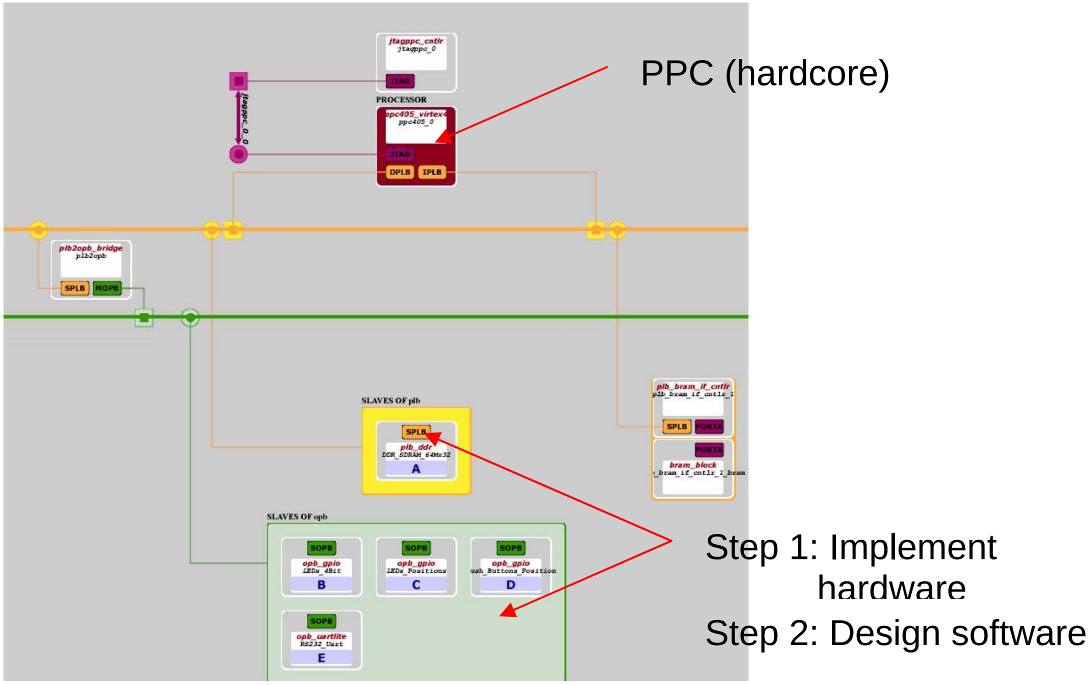


Figure 15. The concept behind implementing a design on the PPC

The PPC already exists on the FX core. We will just implement various hardware in the FPGA fabric on the FX core and then write software (in C) to glue the hardware to the PPC. For more details, refer to [5].

Double click on the 

Xilinx Platform Studio (XPS) icon to start EDK 9.1. Figure 16 should pop up.

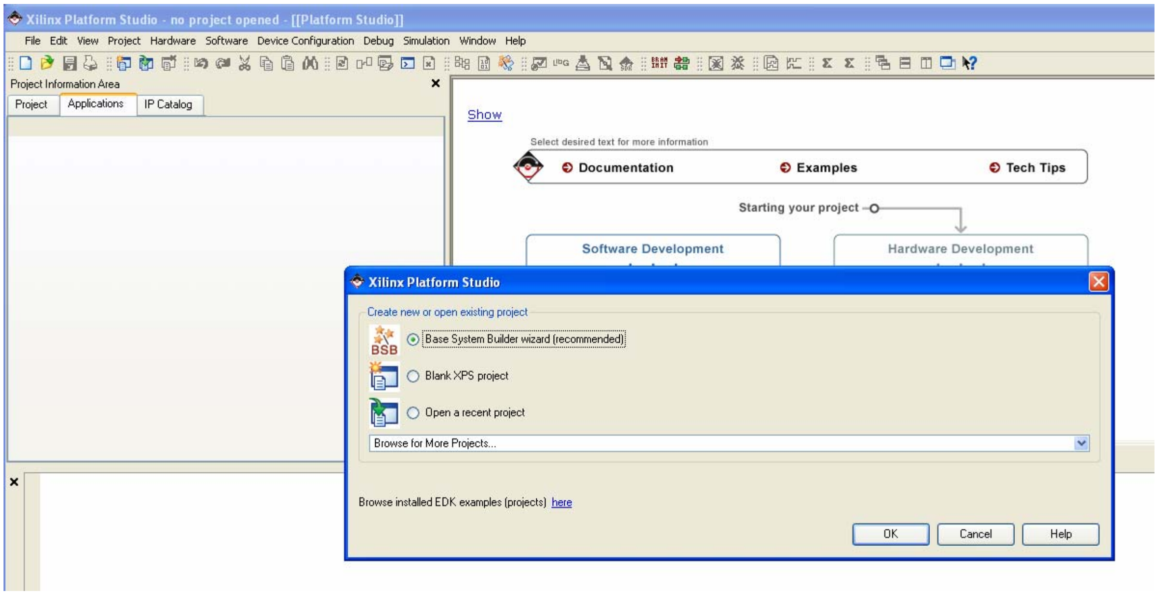


Figure 16. EDK 9.1 startup screen showing the base system builder

Step 1: I highly recommend using the Base System Builder to build the project skeleton. Select OK to build a new project.

Step 2: Create a new directory for the project called LED_blink_ml403_ppc, as shown in Figure 17.

Figure 17. A new project created using the base system builder wizard

Step 3: In the Base System Builder Welcome Screen, select “Create a New Design”.

Step 4: Next, select the board vendor as Xilinx and Board name as the ML403 Evaluation Platform.

Step 5: Select PowerPC as your processor (the MicroBlaze is a soft-core). Pay careful attention to the block diagram in this box.

Step 6: Make sure all the clock frequencies are 100 MHz (refer to figure 18):

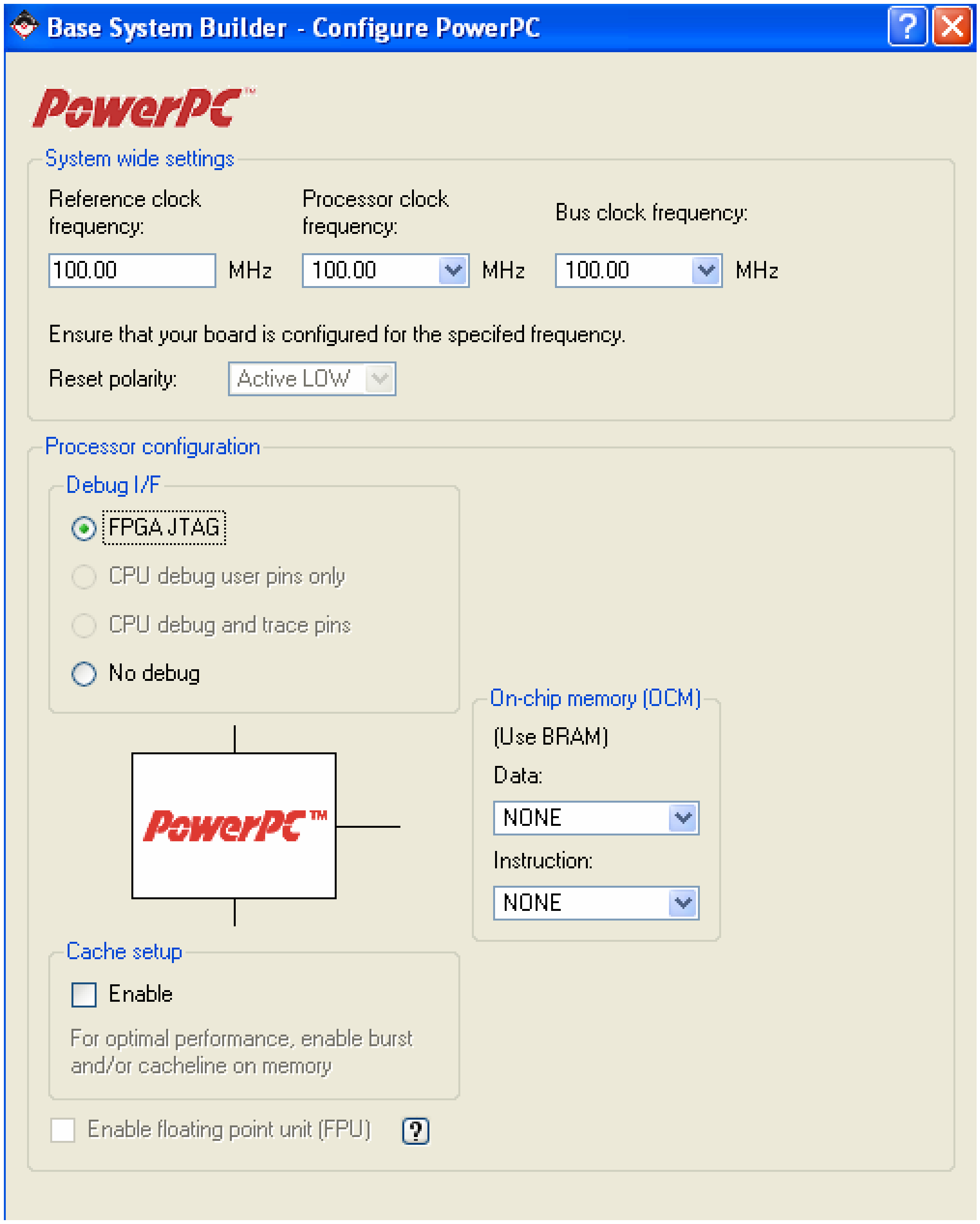


Figure 18. Configuring the PPC

Step 7: Refer to figures 19 configuring the first set of IO interfaces. Select DDR_SDRAM_32M for set 2, but for the next two, deselect everything. Basically you deselect everything but the LED interface, the UART (Universal Asynchronous Receiver Transmitter, you need the UART for console debugging) and the DDR SDRAM.

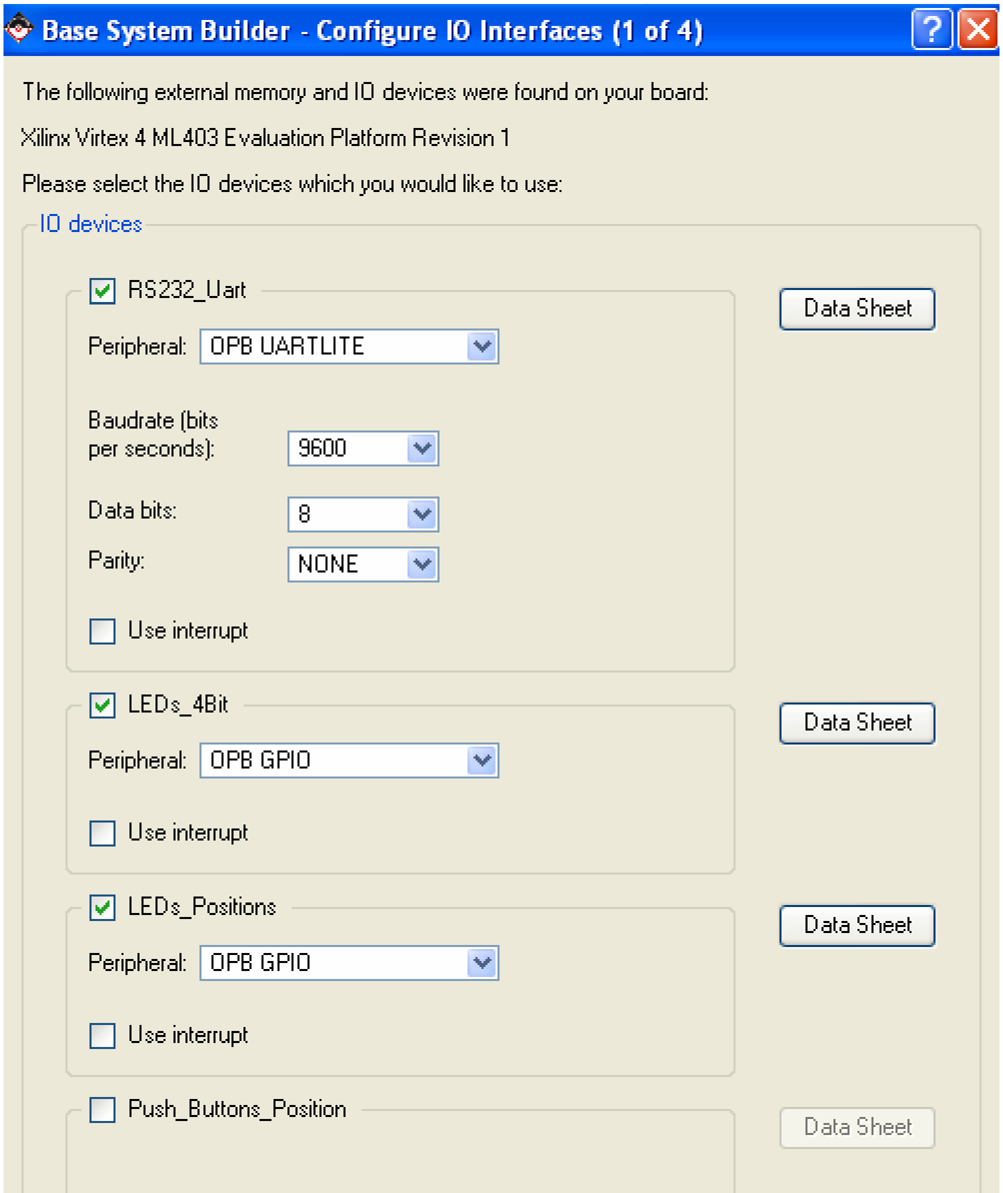


Figure 19. Select the LEDs

Step 8: Select 64 KB for the BRAM. **Make sure that you select a BRAM, PPC will NOT work without BRAM [7].** Click Next to Proceed to Software Setup.

Step 9: Reconfigure the software setup to match figure 20.

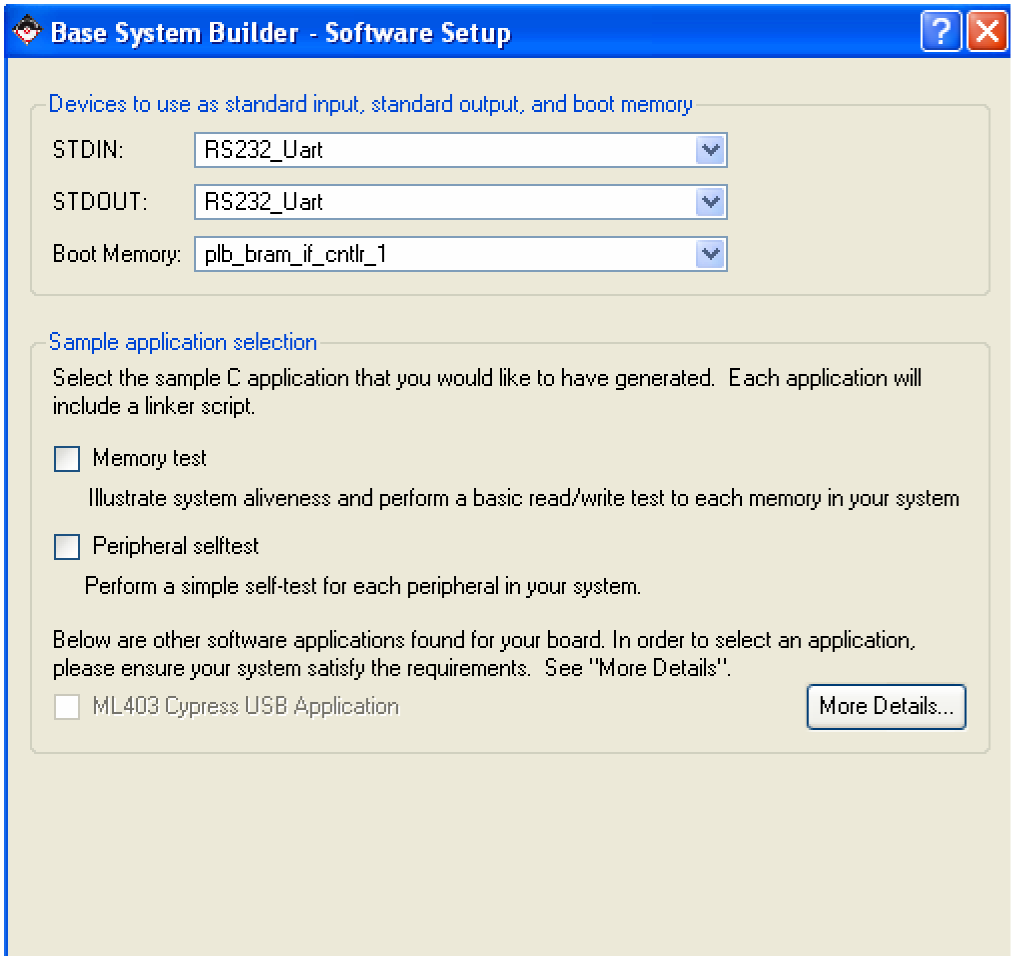


Figure 20. Deselect the Memory test and the Peripheral selftest

Step 10: Click Generate to get the code and Finish to open the project in XPS. Figure 21 shows the result.

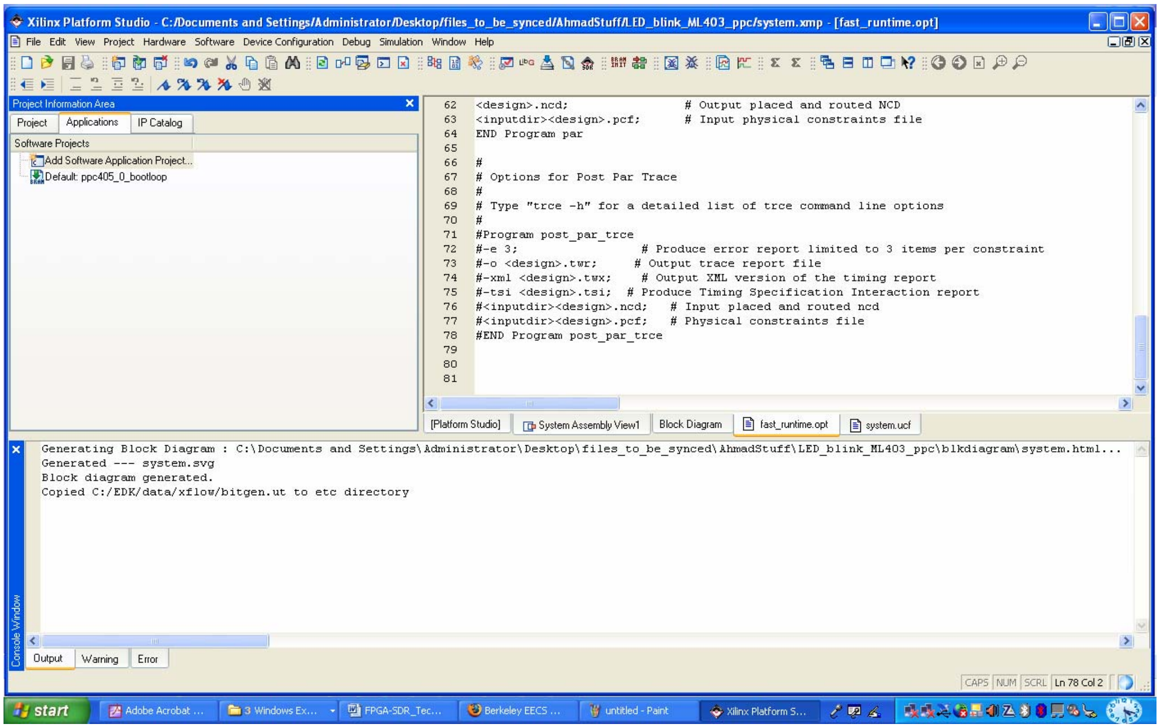


Figure 21. XPS is ready to go!

Step 11: The careful reader would have noticed a difference between Figure 21 and their XPS window. I have opened two files: fast_runtime.opt and system.ucf. To get to these files click on the Project in the Project Information Area and then double click on the two files. The system.ucf file is similar to the one we created in the ISE project.

The fast_runtime.opt is more interesting. When I compiled the project for the first time, I always get an error from the Post Place and Route program: unexpected termination, halting. Since this error message is not very informative, I decided to comment out the post place and route program. Lo and behold! My program compiled and worked on the board. However I don't know the implications of commenting out the Post Place and Route program. I have contacted Xilinx, still waiting for a response. Please email me if you have an explanation or you found an alternate solution. But for the time being, **make sure you comment out the Post Place and Route program in fast_runtime.opt!**

Step 12: The next step is to create the LED source code. Again the idea is we write a "C" program to interface to the LEDs through the GPIO bus.

Finding the various libraries required to program the PPC requires a little bit of mouse clicking:

Step 12 a: Left-click on Help → EDK Online Documentation

Step 12 b: Left-click on IP Reference (refer to figure 22).

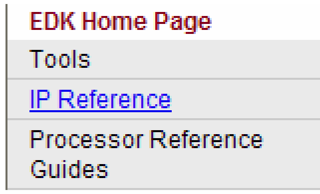


Figure 22. Select IP Reference (underlined above)

Step 12 c: Left-click on Driver Reference Guide (refer to figure 23).

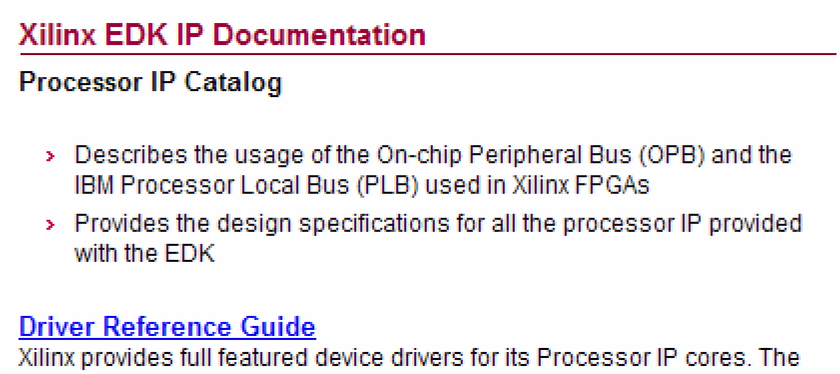


Figure 23. Select Driver Reference Guide (underlined above)

Step 12 d: Left-click on Links (refer to figure 24).

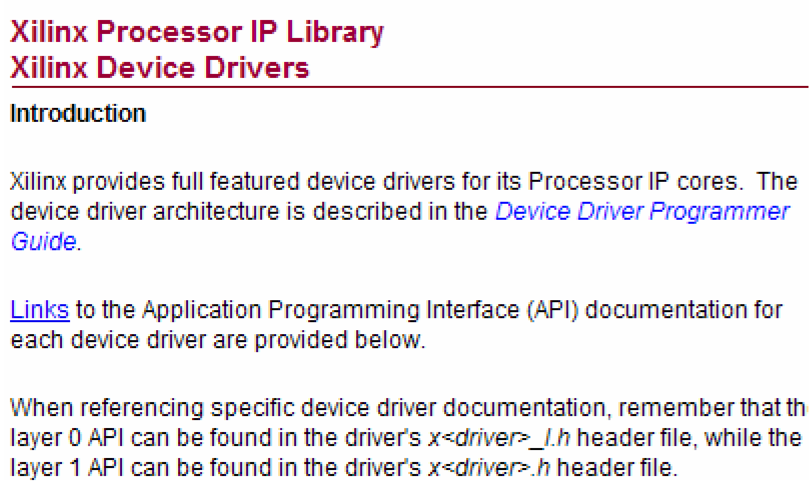


Figure 24. Select Links (underlined above)

Step 12 e: Left-click on Driver API links

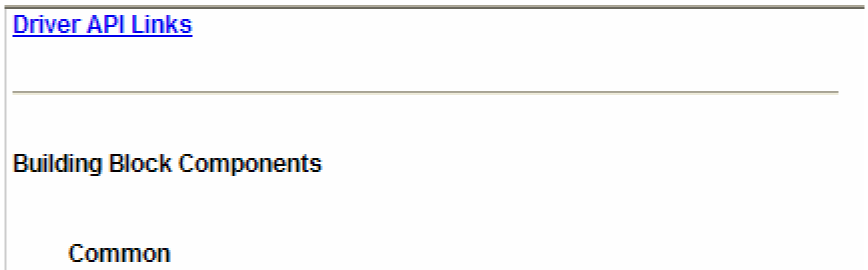


Figure 25. The Driver API links gives you a list of all the APIs

Step 12 f: Scroll down the API list and select GPIO version 2.01a (refer to figure 27).

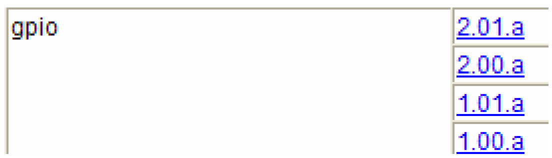


Figure 26. Select 2.01a for GPIO

This will give you the API description for GPIO. The program code for blinking the LED is given below.

```
#include "xgpio.h"
#include "xparameters.h"
#include "xcache_l.h"
#include "xtime_l.h"

int main(void)
{
XGpio gp_out;
int i = 0;
int j = 0;

        XGpio_Initialize(&gp_out, XPAR_LEDS_4BIT_DEVICE_ID);
        XGpio_SetDataDirection (&gp_out, 1, 0x00);

        while (1)
                {
                j = (j + 1) % 256;

                //write the value of j to the LED's
                XGpio_DiscreteWrite (&gp_out, 1, j);
                //software delay loop for pause
                for (i=0; i<400000; i++);
                }
}
```

Create a new directory called code in your project and create a new file with the text above. You can name the file anything you want, I have ledApp.c

Step 13: We need to create space for our file in the project. To do this, double-click on Add Software Application Project under the Applications in the Project Information Area. Refer to figure 27. Name your project led_blink and select OK.

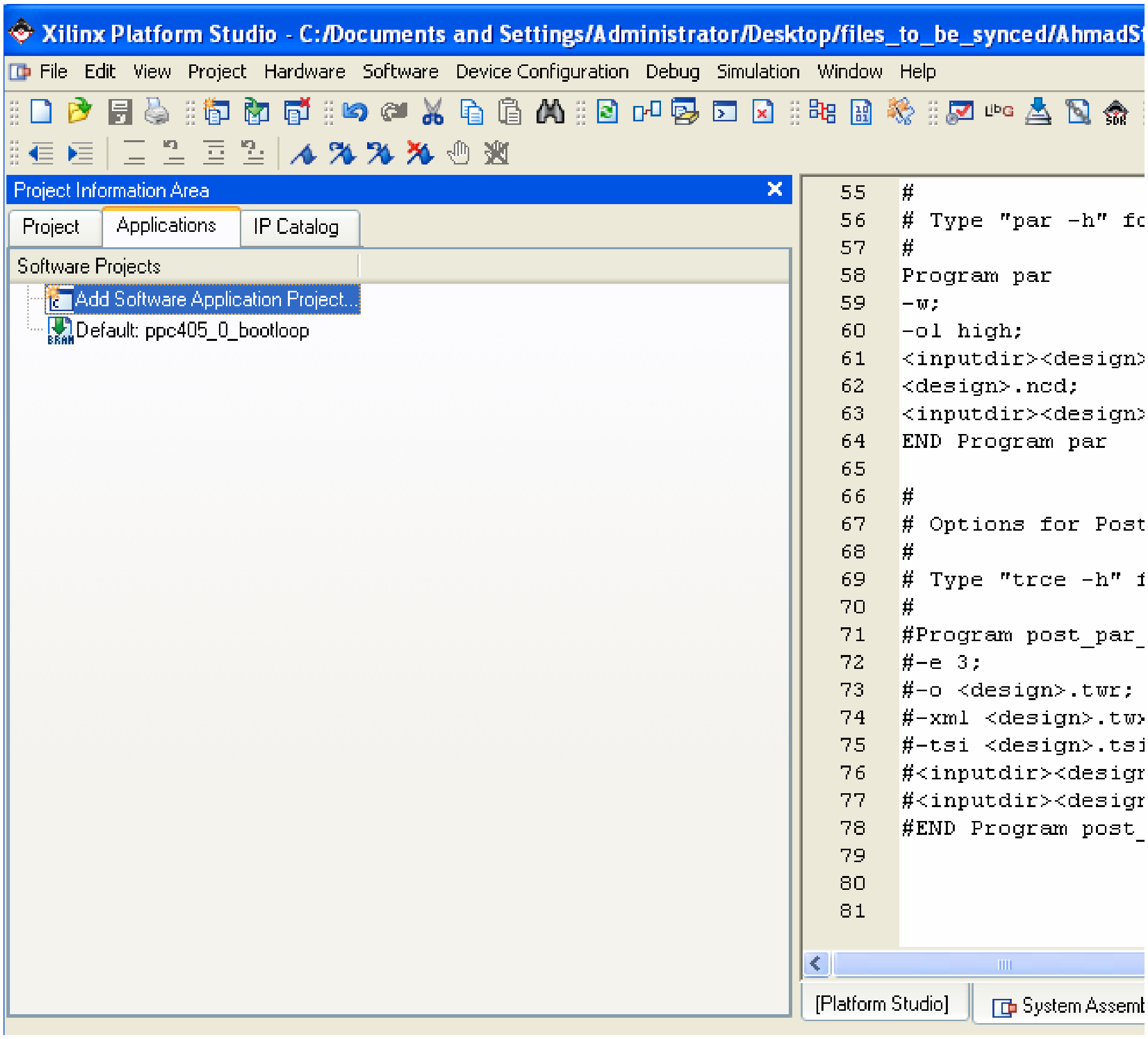


Figure 27. Adding our LED blink to the project

By default our program will be initialized in BRAM. This means the code will be stored in FPGA memory. **Make sure the bootloop program is initialized to BRAM. This keeps the processor in a known state after we load the hardware (the UART interface, the SDRAM interface and GPIO Bus) and before we load the software (our LED blink program).** Right click on “Default: ppc405_0_bootloop” and make sure that “Mark to Initialize BRAMs is selected”.

Step 14: Now the hardware portion of our project is configured, we need to download it to our FPGA board. Click on Device Configuration (in the menu bar) and then Download Bitstream.

Once the FPGA is configured, the DONE LED (above the Compact Flash card on the ML403) will light up. Now you will download our software onto the board.

Step 15: First, add our source file. Right click on Sources and select Add Existing File. Refer to figure 28.

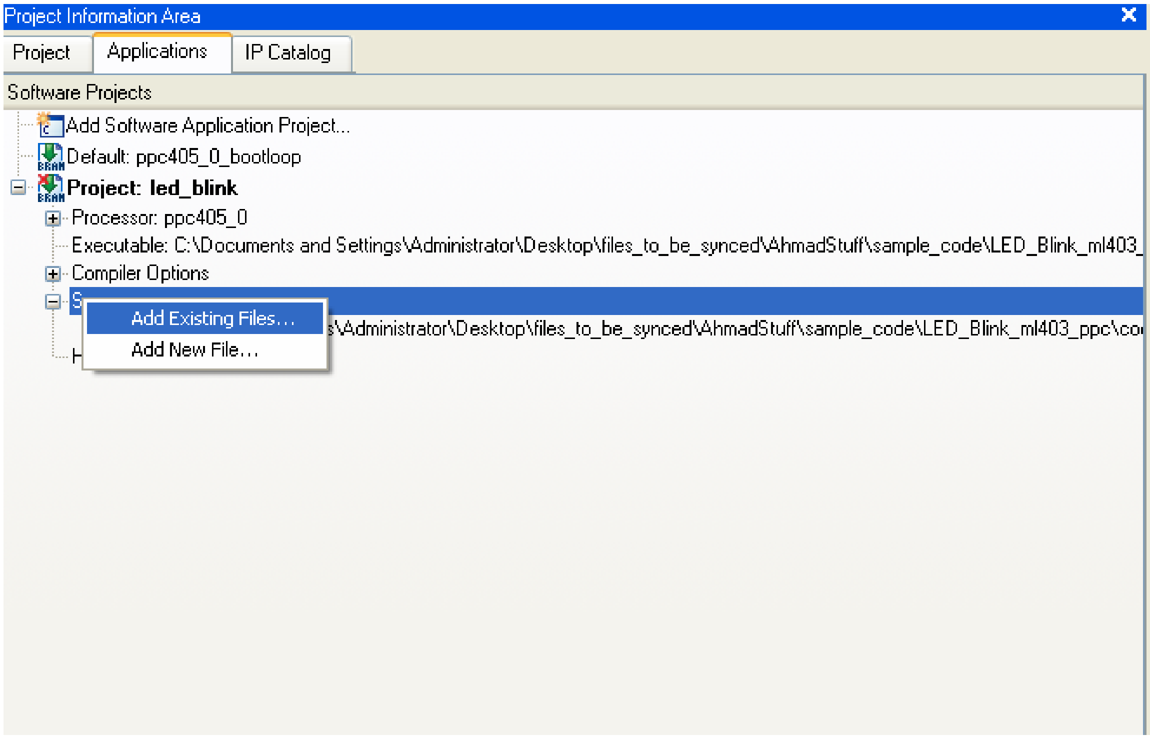


Figure 28. Adding an existing file to the project

Step 16: Next, Left-click on the "+" next to the Compiler Options and Double-click "Linker Script:" We use a linker script to tell the compiler (the linker, technically) where to put the software in memory. The generate linker script dialog shows many different sections of the program and where they will be put in memory. You don't have to understand what these sections mean, just understand that they represent where parts of the program are stored in memory [7].

Refer to figure 29 for the memory options. **THE BOOT PARTITIONS SHOULD BE LEFT IN BRAM [7].**

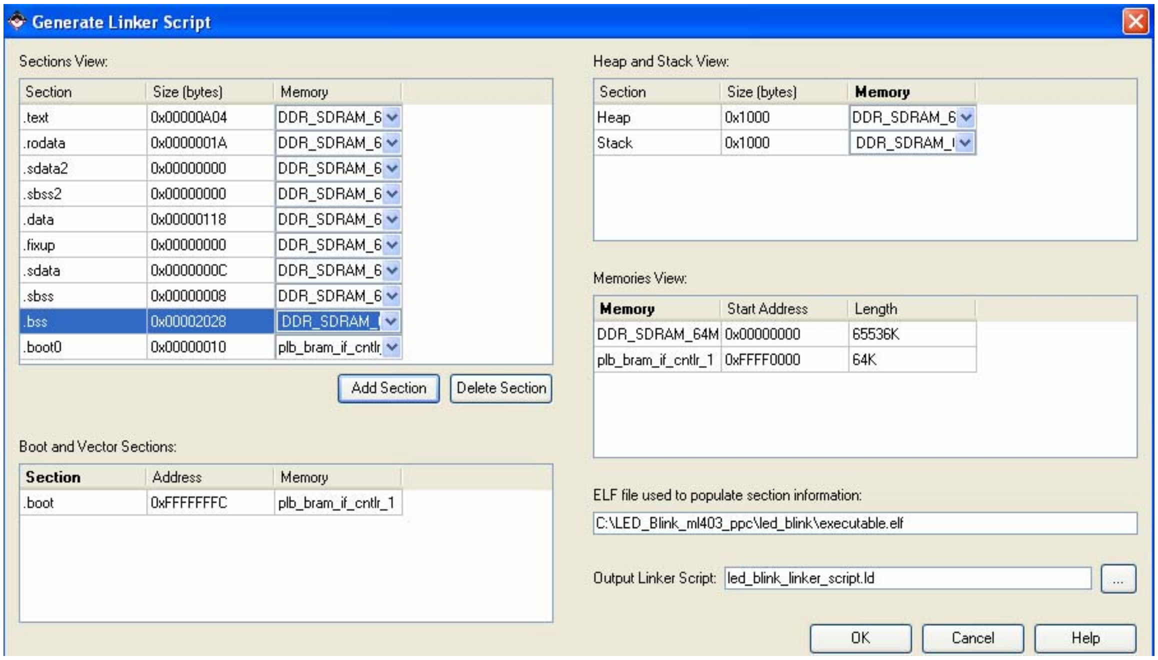


Figure 29. Linker options

Step 17: Before we build the program, we will disable optimizations because they can make debugging difficult. Double-click on “Project: led_blink” and find the “Optimization” tab. Change the “Optimization Level” to “No Optimization”. Also make sure that in the Debug Options section, the Create symbols for debugging checkbox is selected [7].

Step 18: Select Software and then select Build All User Applications in the menu bar.

Step 19: Left-click Debug in the menu bar and Select Launch XMD. Figure 30 will pop up.

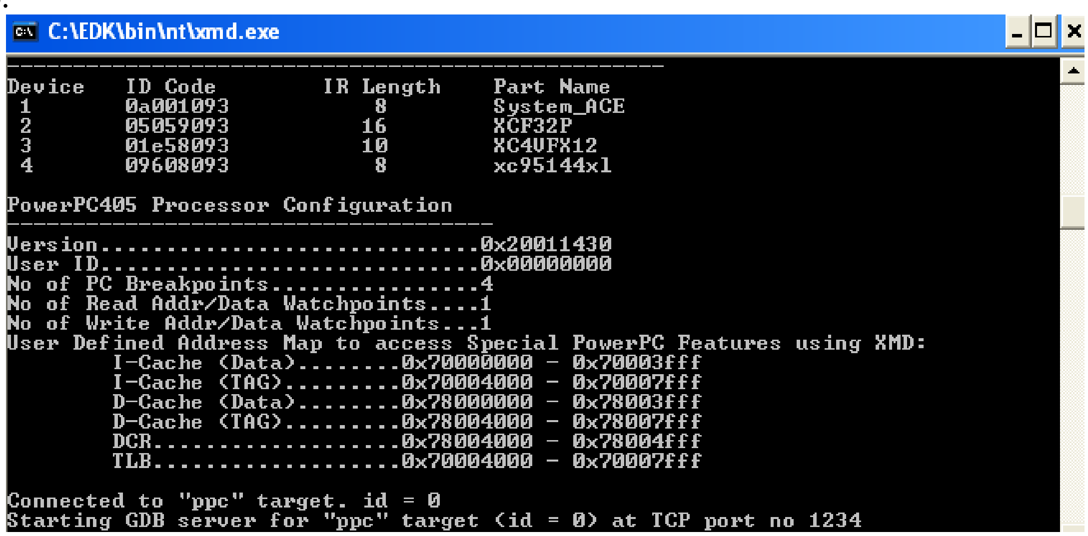


Figure 30. XMD is up and running

Step 20: Type: "dow led_blink/executable.elf" and type "con" to execute the program. The LEDs should blink slowly!

4. System Generator for DSP – Programming FPGAs using Simulink

Xilinx's User's Guide for System Generator [6] is the best reference for learning how to use System Generator. It is a free download (**16 MB**).

5. References

1. *http://www.fpga4fun.com* (last accessed 05/17/07)
2. *Advanced Digital Design with the Verilog HDL*. Ciletti, Michael D. Xilinx Design Series, 2003.
3. *Xilinx Synthesis Technology (XST) User Guide.* pp. 5-10 – 5-12. Online (last accessed 05/18/07): http://toolbox.xilinx.com/docsan/xilinx5/pdf/docs/xst/xst.pdf
4. *ML401/ML402/ML403 Evaluation Platform.* Online (last accessed 05/18/07): http://www.xilinx.com/bvdocs/userguides/ug080.pdf
5. *Embedded Systems Tools User's Manual*. Online (last accessed 05/22/07): http://www.xilinx.com/ise/embedded/est_rm.pdf
6. *Xilinx System Generator User's Guide*. Online (last accessed 05/23/07): http://www.xilinx.com/support/sw_manuals/sysgen_ug.pdf
7. *Introduction to Xilinx Platform Studio.* Iowa State University. Online (last accessed 05/23/07): http://class.ece.iastate.edu/cpre488/Labs/Lab_1/CPRE488_LAB01.pdf

6. Acknowledgements

First off, many thanks to Prof. Ahmad Bahai for find the funding and letting me work on this project. Prof. Pravin Varaiya, Dr. Carl Chun, Ali Djabbari and Dr. Wei Ma from National Semiconductor have been very helpful and continue to help me in this project. .

Raffi Selvian debugged one of the very first incarnations of our installation. Ian Tan helped me countless times with FPGA tips! Sophie, Maryam and Prashanth provided valuable feedback.

Last but not the least, Ferenc Kovac for mentoring me and helping us with the ML403 and tool setups.